UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

WEB SERVICE COMPOSITION VIA SEMANTIC MATCHING OF INTERACTION SPECIFICATIONS

Fausto Giunchiglia, Fiona McNeill and Mikalai Yatskevich

November 2006

Technical Report # DIT-06-080

Web Service Composition via Semantic Matching of Interaction Specifications

Fausto Giunchiglia
Dept. of Information and Communication Technology
University of Trento
38050 Povo, Trento, Italy
fausto@dit.unitn.it

Fiona McNeill
School of Informatics
University of Edinburgh
EH8 9LE, Scotland
f.j.mcneill@ed.ac.uk

Mikalai Yatskevich
Dept. of Information and Communication Technology
University of Trento
38050 Povo, Trento, Italy
yatskevi@dit.unitn.it

ABSTRACT

Web service composition is a difficult and important requirement for the success of the Semantic Web. In this paper, we present a method of facilitating automated service composition through the declarative specification of potential interactions between them (or workflows) and the provision of semantic matching support to allow the automated interpretation of the interaction specifications. The key idea is that matching allows us, at run time, to find the correspondences among independently defined heterogeneous web service descriptions. Our approach extends existing methods to allow interaction models to convey relevant matching information and develops novel semantic matching techniques which allow for the interpretation of logical terms. The preliminary evaluation results show high efficiency and effectiveness of the proposed matching techniques.

1. INTRODUCTION

The automation of Web service interaction and composition is a major goal of the (Semantic) Web and its achievement would have profound affects on the potential of machine intercommunication. However, this is a difficult goal to achieve and a great deal of work has been invested in its pursuit without, so far, any very definite success. The difficulties surrounding this problem are grounded in the fact that it is very hard to have successful and predictable interactions between systems or services with discordant semantics, and yet it is also impractical and limiting to force a single view of semantics on an unlimited number of potential services; such an approach could only be feasible within a small, controlled domain. The tension between the need for open, unrestricted semantics and the need for semantic uniformity has driven much of the research on the (Semantic) Web.

In this paper, we present a novel approach to this problem. We facilitate web service composition through the use of declarative specifications of the interactions in which the services may be involved and then provide semantic matching techniques so that web services can interpret these interaction specifications without adhering to any specific form of knowledge representation. The interaction specifications, referred to as *interaction models*, are shared between services or peers on the network and can be discovered prior to or during run-time. This idea of developing declarative interaction models to enable unforeseen interactions during run-time is not new; rather, we are building on work posited in [4] in which the philosophy and practice of controlling interactions between disparate services are defined through the use of the Lightweight Coordination Calculus (LCC) language. Using these interaction models, the problem of discordant semantics is thus reduced to the semantics directly concerning the interaction in question; everything else can be safely ignored. These interaction models can be considered as contracts describing the semantics necessary for the integrity of specific interactions. The sharing of these interaction models allows services, or peers, to participate with other, unknown, peers in interactions that had not previously been anticipated. In order to understand what the interaction means and what it involves, it is necessary for each peer to interpret the terms in the interaction with respect to its own knowledge, thereby solving semantic heterogeneity problem arising between the part of its knowledge and the knowledge contained in the interaction model. Additionally, in order for the interaction to be successful, all peers involved should interpret the knowledge in the same way. Our approach, therefore, does not remove the problem of semantic heterogeneity entirely; rather, it substantially limits the scope of this problem and provides a specific context in which it has to be solved. This limitation of scope allows us to build feasible run-time semantic matching techniques.

We address the semantic heterogeneity problem through

- the extension of the LCC language to allow for contextual information to be provided to the matching process.
- the applications of novel semantic matching techniques intended to match first order logic terms that express both the constraints to be interpreted by a peer and the peer capabilities;

Context [10] thus becomes an object that can be circulated, minimizing the effects of semantic heterogeneity. The application of our semantic matching techniques, aided by contextual information, in place of unification, extends the applicability of these techniques from a closed to an open domain. We do not limit this notion of context and allow for the possibility of multiple contexts. Any information that might be thought pertinent to the matching process can be included as context. In this paper, we discuss a few examples of what this context may be but, in the general case, the context is not limited to such examples. Our semantic matching techniques build on previous work done in this field [8, 9] but extend this work in a new direction to enable the semantic matching of logical terms. Preliminary evaluation results support viability of our approach.

The rest of the paper is organized as follows. Section 2 uses a motivating scenario to outline the lifecycle of the interaction process. Section 3 introduces the LCC language and describes the extensions we have made to it. Section 4 outlines the matching

WWW2007, May 8–12, 2007, Banff, Canada.
.

techniques. Section 5 presents some preliminary evaluation results. Section 6 explains our work in relation to other work in the field and explains the additional contributions we have made. Section 7 summarizes the paper and draws conclusions.

2. THE APPROACH

The OpenKnowledge system [22] is designed to allow peers on a network to search for interaction models that describe the interaction which they wish to initiate and to locate other peers to play the necessary roles in the interaction (with the initiating peer usually playing at least one role). Neither the interaction model nor the other peers need to be known before run-time, though this can be the case if desired. The work in this paper facilitates the automatic interpretation of these interaction models so that the initiating peer can determine whether the interaction model is really appropriate for its needs and peers potentially suitable for playing other roles can determine how they are able to fulfil the roles and whether the consequences of that role are compatible with their goals.

The lifecycle of an interaction is as follows [22]:

- **STEP 1** The initiating peer must first locate an appropriate interaction model that results in the goals it wishes to satisfy which can either be already known to it can be found via the discovery service. A discovery service, using lightweight matching techniques, such as keyword matching, returns interaction models satisfying the request. The description of a discovery service is out of scope of this paper. We refer an interested reader to [12] as one of the possible implementations. Once these potentially suitable interaction models have been located, semantic matching is used to determine if they are appropriate for the task in hand.
- **STEP 2** The discovery service (discussed in Step 1) will find potentially suitable peers through matching the role description in the interaction model with the descriptions that peers give of their capabilities;
- **STEP 3** Potentially suitable peers are contacted and, if they are available and willing, will be sent a copy of the interaction model;
- **STEP 4** Each peer will perform semantic matching to interpret the requirements and effects of the interaction. If they are happy with the consequences of the role and are able to fulfil the constraints, they will return this information.
- **STEP 5** The suitable peers are ranked according to the trust values associated with them, and, in the advanced case of approximate matching, their matching scores. This trust value may come from the results of previous interactions with the peers and is not discussed in this paper. The highest ranked peers are approached to play the roles in the interaction model.

In this paper we concentrate on Steps 4, which is when the matching of peer capabilities with the role constraints is performed.

EXAMPLE 1. *(Wine selling on the web) James is a wine merchant and regularly buys online from known distributors with whom he has an established relationship. In addition to this, he is keen to find good deals and special offers from unknown distributors. He sells his goods to customers in his shop and online and has different methods of doing this: for example, customers may want to request a particular kind of wine or may wish to provide some constraints such as price, color, etc., and would accept recommendations on that basis. James is keen to advertise himself as a wine seller to as many potential customers as possible and to be accommodating in his interactions with them so as to encourage custom. He also wishes to actively search for suppliers to ensure that he is always able to fulfil his orders and replenish his stock and to ensure he keeps his costs to a minimum.*

It is advantageous that as much as possible of this is done automatically. James will therefore have a peer representing him which will become involved in an interaction either by initiating it or by being approached by another peer who wished to initiate an interaction. Here, we list the necessary steps for James's peer to initiate an interaction, and describe the behavior of the peers who are approached to become involved in the interaction.

If James wishes to proactively sell wine, his peer will first select an interaction model either from ones it knows or through discovery, locate a peer who wishes and is able to play the role of being a customer in a wine selling context and then initiate the interaction.

3. EXPRESSING INTERACTIONS

In this section, we briefly describe the lightweight coordination calculus (LCC) on which our interaction language is based; further details can be found in [4]. We then describe the extensions we have made to this language to facilitate semantic matching.

3.1 LCC

LCC is designed to allow a simple, lightweight method of developing declarative descriptions of interactions. Figure 3.1 defines the syntax of LCC. An interaction model written in LCC consists of a set of clauses, each describing a role in the interaction. We list below the essential components of a role descriptor.

- Peer identifiers for peers involved in the interaction, which consist of the predicate a, which has arguments detailing the role type of the relevant peer and the particular identifier of the peer playing that role in the current interaction.
- An ordered sequence of activities that the peer must perform in the execution of the role. These activities may be:
  - message passing. Messages are either outgoing to another peer playing a given role defined by the interaction model ($\Rightarrow$), or incoming from another peer playing such a role ($\Leftarrow$). If the message is incoming, the peer cannot proceed further with the role until it has received it.
  - changing to another role (whose description must also be contained in this interaction model). When a peer changes to another role the extra arguments attached to the role identifier might be necessary: information that has been discovered during the performance of the role so far may need to be passed to the new role.
- A sequence operator (*'then'*) and a choice operator (*'or'*) to connect activities.
- Constraints, which are attached to activities and describe under what circumstances they may be performed under. Constraints can be attached to outgoing messages or to role changes: incoming messages can always be received by peers and thus cannot have constraints on them. Additionally, constraints may be used to represent $consequences$ of message passing. Throughout this paper, we will tend to refer to both constraints and consequences as constraints, since they are

$Model$	:=	$Clause, ...$
$Clause$	:=	$Role :: Def$
$Role$	:=	$a(Type, Id)$
Def	:=	$Role \mid Message \mid Def\ then\ Def \mid Def\ or\ Def$
$Message$	:=	$M \Rightarrow Role \mid M \Rightarrow Role \leftarrow C \mid M \Leftarrow Role \mid C \leftarrow M \Leftarrow Role$
C	:=	$Constant \mid P(Term, ...) \mid \neg C \mid C \wedge C \mid C \vee C$
$Type$	:=	$Term$
Id	:=	$Constant \mid Variable$
M	:=	$Term$
$Term$	:=	$Constant \mid Variable \mid P(Term, ...)$
P	:=	$Constant$
$Constant$	:=	lower case character sequence or number
$Variable$	:=	upper case character sequence or number

Figure 1: LCC syntax

syntactically identical and the issues surrounding their interpretation are the same. We only distinguish when this is necessary for clarity: for example, when we are discussing a specific example.

The constraints are the aspects of an interaction model in which the meaning of that interaction model is grounded and a peer cannot understand whether it can perform the role or what the effect of performing the role will be to it unless it can interpret these constraints with respect to its own knowledge. LCC does not enforce any commitment to the system of logic through which constraints are solved, so different peers can use different constraint solvers (including human intervention). The constraints in LCC are written in first-order CNF.

EXAMPLE 2. *(A Wine Selling Interaction Model) In this section, we describe one of the interaction models used by the wine merchant, described in Figure 2. This interaction model could be one already known to James's peer and may be stored by him in a manner facilitated by the OpenKnowledge interface; for example, Figure 4 describes a way in which James's interaction models may be stored. In such a case, the contextual information derived from this storage hierarchy could be useful during matching. However, an interaction model discovered during run-time may have no such information attached to it, and even if the interaction model is already known, James may not choose to organize his interaction models so carefully. The improvement this gives to the quality of the semantic matching means it is in the user's interests to do this but it is not obligatory. The interaction model describes, for each of the two roles in the interaction ($customer$ and $wine\_merchant$), the messages that are passed between them during this interaction. Since there are only two roles in this interaction, the message passing described in them is symmetric. In addition, the constraints and consequences of the message passing is described.*

The message passing is indicated by the double arrows ($\Rightarrow, \Leftarrow$): the messages that are to be sent are shown to left of these arrows and the role from whom they are expected or to which they are to be sent ($wine\_merchant$ or $customer$) are shown on the right. Since each role description is written from the point of view of a peer playing the role, it is only necessary to mention a single role in the message passing; the other role is implicit. To the left-hand side of the message are the constraints: for example, $choose\_wine$ and $recommend$. These can only be attached to messages to be sent as there are never constraints on receiving messages. Either sent or received messages can, in some cases, have consequences instead. In Figure 2 we can see that there is a consequence, $owns$, to the $customer$ on the receiving the message $sold$. Sending the message $sold$ also has consequences for the $wine\_merchant$. It can be seen from the message passing that the ordering is fully determined. The $wine\_merchant$ cannot proceed with its role until it has received a message from the $customer$, and thus this message from the customer must be the one that initiates the interaction. Every message after that can only be sent after the receipt of the previous message.

First the $choose\_wine$ constraint have to be satisfied by the peer playing the $customer$ role. During the process it discovers the pertinent information that is to be sent in the message. In order to satisfy the constraint, it must interpret it by mapping it to a constraint which it knows how to satisfy, and then run its constraint satisfaction program on it. It then waits to receive a recommendation from the $wine\_merchant$, together with a specific price. It must then attempt to satisfy the $accept(Pr, M)$ predicate, which determines whether the response is acceptable. If the constraint is satisfied, it then waits to receive a message confirming the sale. The consequence of receiving this message is that the $customer$ peer now owns these bottles (there are also likely to be financial consequences but these omitted for the sake of brevity).

Note that in order to understand the interaction model, the peer only need interpret the constraints and consequences. The predicates in the messages are just place holders that allow the identification of the particular message; sensible names for these predicates are used to make things easier for the interaction model designer and for any users who wish to look at it but do not need to be interpreted by the peers.

3.2 Contextual LCC

As described above, each peer must be able to interpret and then satisfy the constraints on the activities in the role it desires to play before it can either assert that it is able to play the role or start the interaction. If there is no prior agreement on what interactions will take place or on the interaction models that describe them and there is no central ontology or definitive method of knowledge representation, then there is no way of ensuring that a peer's way of representing the constraints it can satisfy will match the way in which these constraints are represented in the interaction model. Peers cannot be expected to have a uniform vocabulary, nor can they be expected to represent complex concepts in the same way.

In LCC, the matching of a constraint that a peer knows how to satisfy (or at least, knows how to determine whether it can satisfy) and a constraint which it must satisfy in a particular interaction model is done through unification; if the representation and vocabulary used is not the same, matching will fail, and thus it is only applicable in a closed domain. In an open domain, peers will find themselves unable to understand constraints even on roles that they have the ability to play.

We propose to extend LCC to facilitate semantic matching on the constraints by adding contextual information to interaction models. It is possible to perform semantic matching on constraints even without any contextual information; however, this can make it difficult to find good semantic matches.

Another aspect missing from the LCC definition that is important in matching is type information. If type information can be attached to the arguments of constraints then it is much easier to determine how well they map to known constraints. This is especially true since, when the matching is being done on constraints, before the interaction commences, it is likely that most of the arguments of

$a(customer, C)$ ::

$$request(wine(P_1, P_2, R, C, N) \Rightarrow a(wine\_merchant, W) \leftarrow choose\_wine(P_1, P_2, R, C, N)$$
$$recommendation(Pr, M) \Leftarrow a(wine\_merchant, W) \; then$$
$$buy(M, N_2) \Rightarrow a(wine\_merchant, W) \leftarrow accept(Pr, M, N_2) \wedge N_2 \leq N$$
$$owns(M, N_2) \leftarrow \quad sold(M, N_2) \Leftarrow a(wine\_merchant, W)$$

$a(wine\_merchant, W)$ ::

$$request(wine(P_1, P_2, R, C, N) \Leftarrow a(customer, C) \; then$$
$$recommendation(Pr, M) \Rightarrow a(customer, C) \leftarrow recommend(R, C, M, Pr) \wedge Pr > P_1, P_2 \wedge in\_stock(X, M) \wedge X \geq N \wedge price(M, Pr)$$
$$buy(M, N_2) \Leftarrow a(customer, C) \; then$$
$$in\_stock(Y, M) \leftarrow \quad sold(M, N_2) \Rightarrow a(customer, C)$$
$$\wedge Y = X - N$$

Figure 2: Wine buying interaction model

$Constant$	:=	$Context : (Value, Class)$
$Variable$	:=	$Context : (Holder, Class)$
$Class$	:=	$Value \mid Holder$
$Context$	:=	$Value \mid Holder$
$Value$	:=	lower case character sequence or number
$Holder$	:=	upper case character sequence or number or white space

Figure 3: Alterations to LCC syntax required by Contextual LCC

constraints will be variables and thus without this type information it may be impossible to unambiguously determine their meaning.

The changes to the definition that are required in order to replace resolution with semantic matching are minimal. Figure 3.2 details these definitions: the syntax of Contextual LCC is identical to the syntax of LCC with these specified changes. The differences are the way in which $Constants$ and $Variables$ are defined and the addition of four new objects: $Class$, $Context$, $Value$ and $Holder$. Constants and Variables both become complex objects which contain, respectively, a $value$ or a $holder$ (whose definitions are nearly identical to the original definitions of $constant$ and $variable$), and information pertinent to that value or holder: type information (which we refer to here as $class$ to avoid confusion with the $type$ definition that refers to role types) and contextual information. $Class$ information is a value or a holder. $Context$ information is less constrained: anything that defines the context in a term could be used can provide the context, for example, the classification of the interaction model [9], Web directories [1] or even user preferences [25]. $Value$ is defined identically to LCC $constant$; $Holder$ is similar to $Variable$ but, in addition to an upper case character sequence or number being an acceptable input, white space is also acceptable and can be used in the case that an interaction model designer does not know the pertinent information or does not wish to input it. The design of this syntax means that it is easy to add extra attributes to constant and variable definitions should we wish to do so.

Contextual information can be derived from the context in which the interaction model is used. For example, users might store information about their interaction models in a classification such as the one illustrated in Figure 4. The interaction model shown in Figure 2 would be the one stored under $Wine\text{-}Selling\text{-}Recommendation$. This contextual information can be attached to any object in the interaction model and may prove useful in its semantic interpretation.

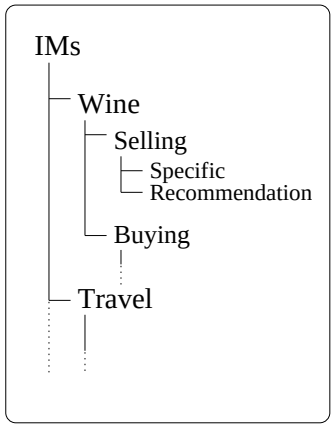


Figure 4: James's interaction models (IM) classification

EXAMPLE 3. *(Marking up the Interaction Model) Figure 5 shows part of the interaction model of Figure 2 with the additional mark-up included. Since the interaction model with all this information explicit is rather unwieldy, we include only a small section.*

Both the type and the context can be extremely useful in interpreting the semantics. For example, the $wine$ context indicates that the variable C, of type $colour$, should be instantiated by red or white; and that the variable M, of type $make$ should be instantiated by some make of wine and not, for example, a make of car.

Notice that predicates, which, in the LCC syntax, are defined as $Constants$, have the same structure as any other constraint: $Context : (Value, Class)$. It may be felt that type information is inappropriate in describing predicates and, if so, this attribute will be instantiated by a white space, as occurs in the type attribute of $choose\_wine$ in Figure 5. However, if the interaction model designer or user wishes to give type information to predicates, he is at liberty to do this.

4. STRUCTURE MATCHING

In order to perform semantic matching between the first-order constraints found in an LCC interaction model, we consider the first-order terms as trees and perform tree matching on them. There are two stages in the matching process:

1. Node matching: solves the semantic heterogeneity problem by considering only labels at nodes and their contextual information inside constants in interaction model and web ser-

$a(customer, C)$ ::
$request(wine(P_1, P_2, R, C, D, N) \Rightarrow a(wine\_merchant, W) \leftarrow (Cxt : choose\_wine,))$ $(Cxt : (P_1, maxium\_price), Cxt : (P_2, minimum\_price), Cxt : (R, region), Cxt : (C, colour), Cxt : (N, number\_of\_bottles)$

$Cxt = (wine\text{-}selling\text{-}recommendation)$

Figure 5: Semantic markup of interaction model

vice descriptors.

2. Tree matching: exploits the results of the node matching and the structure of the term to find an overall match between the terms in a web service description and in interaction model.

4.1 Node matching

Semantic matching, as from [8] is based on the 2 key notions:

- *Concept of a label*, which is a logical formula encoding the meaning of a label;
- Concept of a node, which is a logical formula that encodes the meaning of a node, given that it has a certain label and it is in certain position in the term tree and in the IM classification (see Figure 4 for example).

We say that two nodes n_1 and n_2 in the trees T_1 and T_2 (semantically) match iff the formula "$c@n_1$ *iff* $c@n_2$" holds given the available background knowledge, where $c@n_1$ and $c@n_2$ are the concepts at nodes of n_1 and n_2 respectively.

The semantic node matching algorithm, as introduced in [9], takes as input two term trees and computes as output a set of correspondences holding among the nodes in the trees in four macro steps:

- *Step 1: for all labels L in two trees, compute concepts of labels, C_L.* Step 1 is concerned with automatic translation of ambiguous natural language labels taken from the term tree elements into an internal logical language with boolean semantics (see [9] for more detail). The process involves tokenization, lemmatization, querying the Oracle (such as WordNet [19]) in order to determine the label senses, and, finally, the complex concept construction. The last step is concerned with interpretation of certain natural language labels as the logical connectives (for example both natural language *and* and *or* are translated into disjunction) and word sense disambiguation (see [17] for more detail). Thus, for example, the concept of label *Number of bottles* is computed as $C_{Number\ of\ bottles} = C_{Number} \sqcap C_{bottles}$, where $C_{bottles} = \langle bottle, senses_{WN} \# 4 \rangle$ is taken to be the union of four WordNet senses, and similarly for *number*.
- *Step 2: for all nodes N in two trees, compute concepts at nodes, C_N.* During Step 2 we analyze the meaning of the positions that the labels of nodes have in a tree. Term trees are hierarchical structures where the path from the root to a node uniquely identifies that node (and also its meaning). We define the logical formula for a concept at node as a conjunction of concepts of labels located in the path from the given node to the root. For example, in the Figure 6, the concept at node for the node Region(C) is computed as follows: $C_{Region(C)} = C_{Wine} \sqcap C_{Champagne} \sqcap C_{Region}$.

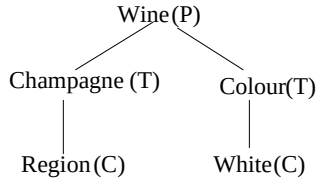
  In order to constrain the set of possible concept at node interpretations the sense filtering techniques are used (see [17, 9] for detailed discussion). The main goal of sense filtering techniques is to filter out irrelevant (for the given matching task) Oracle senses from concepts of labels. For all concepts of labels we collect all their ancestors and descendants. We call them a focus set. Notice that the interaction model itself can be classified in tree like structure (see Figure 4 for example). Therefore focus set is enriched with concept of labels of the interaction model and its ancestors in the IM classification. For example, as from Figures 4 and 6, for the concept at node $C_{Region(C)}$ the focus set contains the following concepts of labels: C_{Wine}, $C_{Champagne}$, C_{Region} taken from the term tree along with C_{Wine}, $C_{Selling}$ and $C_{Recomendation}$ taken from IM classification. Then, all Oracle senses of atomic concepts of labels from the focus set are compared with the senses of the atomic concepts of labels of the concept at node. If a sense of atomic concept of label is connected by an Oracle relation with the sense taken from the focus set, then all other senses of these atomic concepts of labels are discarded. Therefore, as a result of sense filtering step we have (i) the Oracle senses which are connected with any other Oracle senses in the focus set or (ii) all the Oracle senses otherwise. After this step the meaning of concept of labels is reconciled in respect to the knowledge residing in both the term tree and IM classification structures.
- *Step 3: for all pairs of labels in two trees, compute relations among C_L's.* Step 3 is concerned with acquisition of "world" knowledge. Relations between concepts of labels are computed with the help of a library of element level semantic matchers (see [9] for more detail). These matchers take as input two concepts of labels and produce as output a semantic relation (e.g., equivalence, more/less general) between them. For example, from WordNet we can derive that region and area are synonyms, and therefore, $C_{region} = C_{area}$.
- *Step 4: for all pairs of nodes in two trees, compute relations among C_N's.* Step 4 is concerned with the computation of the relations between concepts at nodes. This is done by reducing this problem to a propositional satisfiability (SAT) problem and by exploiting state of the art SAT decider [8, 9].

It is important to notice that Step 1 and Step 2 can be done once for all, independently of the specific matching problem. Step 3 and Step 4 can only be done at run time, once the two trees which must be matched have been chosen.

4.2 Tree matching

In order to match first-order terms, we consider them as trees. Thus a constraint such as $wine(Cxt : (champagne, region), Cxt : (white, colour)$ would be represented as shown in Figure 6, where Cxt is left implicit.

In order to satisfy a set of constraints in a message in an interaction model, it is necessary to satisfy at least one constraint of every disjunction in the CNF. In a more complex situation, we might



P = predicate; T = type; C = constant

Figure 6: Constraint $wine(champagne, white)$ expressed as a tree

match a single constraint to many constraints, or match many-to-many; however, we only consider one-to-one matching in this paper.

Semantic node matching is done prior to the tree matching process, and the results of this are used to determine which nodes in the trees correspond to each other, and, when extended to deal with approximate mapping, how strong this correspondence is. Semantic tree matching is thus the combination of the results of semantic node matching with techniques that take into account the structure of the term. It must determine not only whether the objects used are the same or similar but whether they are organized in the same manner. This organization of the terms encodes important semantic information about how they relate to one another, and the semantic tree matching techniques determine whether these relationship are the same or similar between apparently different constraints.

In order to participate in this process, peers must have representations of constrains they know how to satisfy written in the same manner as the LCC constraints. This does not enforce any control on the representation of their knowledge base in general, it merely requires that peers are able represent their abilities in this manner and that they can satisfy such constraints. These known constraints are what the matching process must refer to.

We say that two trees T_1 and T_2 match iff for any node n_{11} in T_1 there is a node n_{21} in T_2 such that

- n_{11} semantically matches n_{21};
- n_{11} and n_{21} reside on the same depth in T_1 and T_2 respectively;
- all ancestors of n_{11} are semantically matched to the ancestors of n_{21};

At this stage, we assume that the problem of semantic node matching has been dealt with and can be called as a subprocess of the semantic structure matching, the details of which are discussed in Section 4.1.

The pseudo code in Figure 7 illustrates an algorithm for exact structure matching. **exactStructureMatch** takes two trees of nodes (i.e., tree representation of LCC terms) *source* and *target* as an input. Here and throughout the paper we assume that the source tree is derived from an interaction model constraint and the target tree represents the term derived from the peer capability description. **exactStructureMatch** returns an array of *MappingElements* holding between the nodes of the trees if there is an exact match between them and null otherwise. Array of *MappingElements result* is created (line 12) and filled by **exactTreeMatch** (line 13). **allNodesMapped** checks whether all the nodes of source tree are mapped to the nodes of the target tree (line 14). If this is the case there is an exact structure match between the trees and the set of computed mappings is returned (line 15). **exactTreeMatch** takes

```
1.Node struct of
2.  int nodeId;
3.  String label;
4.  String cLabel;
5.  String cNode;
6.MappingElement struct of
7.  int MappingElementId;
8.  Node source;
9.  Node target;
10. String relation;

11.MappingElement[] exactStructureMatch(Tree of Nodes
                                        source, target)
12. MappingElement[] result;
13. exactTreeMatch(source,target,result);
14. if (allNodesMapped(source,target,result))
15.  return result;
16. else
17.  return null;

18.void exactTreeMatch(Tree of Nodes source,target,
                              MappingElement[] result)
19. Node sourceRoot=getRoot(source);
20. Node targetRoot=getRoot(target);
21. String relation= nodeMatch(sourceRoot,targetRoot);
22. if (relation=="=")
23.  addMapping(result,sourceRoot,targetRoot,"=");
24. Node[] sourceChildren=getChildren(sourceRoot);
25. Node[] targetChildren=getChildren(targetRoot);
26. For each sourceChild in sourceChildren
27.  Tree of Nodes sourceChildSubTree=
                            getSubTree(sourceChild);
28.  For each targetNode in target
29.   Tree of Nodes targetChildSubTree=
                            getSubTree(targetChild);
30.   exactTreeMatch(sourceChildSubTree,
                targetChildSubTree,nodesToMatch);
```

Figure 7: Pseudo Code for Structure Matching Algorithm

two trees of nodes (i.e., tree representation of LCC terms) *source* and *target* and array of MappingElements *result* as an input. It recursively fills *result* with the mappings computed by **nodeMatch** (line 23). **exactTreeMatch** starts from obtaining the roots of *source* and *target* trees (lines 19-20). The semantic relation holding between them is computed by **nodeMatch** (line 21) implementing the node matching algorithm. If the relation is equivalence, the corresponding mapping is saved to *result* array (lines 22-23) and the children of the root nodes are obtained (line 24-25). Finally the loops on *sourceChildren* and *targetChildren* (lines 26-30) allow to call **exactTreeMatch** recursively for all pairs of sub trees rooted at *sourceChildren* and *targetChildren* elements.

The above algorithm is designed to succeed for equivalent terms and to fail otherwise. It expects the trees to have the same depth and for all matching nodes to have the same number of children.

EXAMPLE 4. *(Semantic matching in the Interaction Model) Imagine, for example, that a $wine\_merchant$ peer, acting on James's behalf, sent the interaction model described in Figure 2 to a peer that advertised that it wished to participate in wine buying. This peer would have to evaluate its ability to satisfy the constraints of the role and the desirability of satisfying the consequences of the role before it could play the part. The first constraint it must satisfy is $choose\_wine(P_1, P_2, R, C, N)$, which is illustrated with full mark-up information in Figure 5. Imagine that it knows of no such constraint but that it knows a constraint $pick(C, N, A, P_1, P_2)$. These two constraints, with type information, are represented in Figure 8.*

The node matching algorithm first attempts to match the root node of the left-hand tree, $choose\_wine$, with the root node of the right-hand tree, $pick$. A semantic matcher will find an equiv-

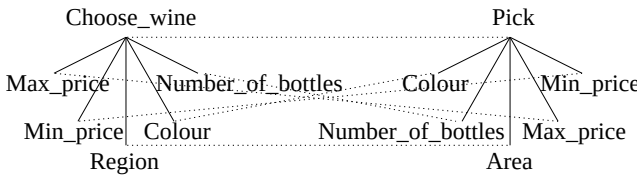


Figure 8: Comparison of two constraints as trees

alence between $choose$ and $pick$, but the $wine$ element has no match in $pick$, thus indicating that perhaps $choose\_wine$ is more specific than $pick$. However, the contextual information can help here, since it gives the information that the terms must be interpreted within a $wine$ context. Thus $pick$ becomes equivalent to $choose\_wine$. Next, a match for Max_price must be found amongst the node at the same level in the right-hand tree. Order is not important here, so a perfect match can be found with the fourth attribute in the right-hand tree. Perfect matches can be found in the same way for all the nodes with the exception of $Region$; however, semantic matching can determine that this is equivalent, in the given context, to $Area$. All nodes in both trees therefore have exact matches, and so an exact match is found between the trees.

4.3 Approximate matching

The matching techniques described in the previous section rely on the semantic content of the constraints known by the peers being identical with that of the constraints found in a particular interaction model. Whilst it is clearly ideal that a peer should perform a role only if it can find constraints it knows that map exactly to the constraints that are required for the performance of the role, this is a heavy demand that may often mean that no suitable peer can be found at all. In practice, we wish to find 'good enough' solutions to queries [11] if perfect answers are no available.

We say that two nodes n_1 and n_2 in the trees T_1 and T_2 approximately match iff $c@n_1$ R $c@n_2$ holds given the available background knowledge, where $c@n_1$ and $c@n_2$ are the concepts at nodes of n_1 and n_2, and where $R \in \{\equiv, \subseteq, \supseteq, \wedge, \perp, not\ related\}$.

We say that two trees T_1 and T_2 match iff there is at least one node n_{11} in T_1 and a node n_{21} in T_2 such that

- n_{11} approximately matches n_{21};
- all ancestors of n_{11} are approximately matched to the ancestors of n_{21};

The key difference between exact and approximate match is in the fact that in the latter case we allow mismatches both on the node and structure level.

The pseudo code in Figure 9 illustrates approximate structure matching algorithm. In contrast to **exactStructureMatch** presented in Figure 7 **approximateStructureMatch** takes as an input not only *source* and *target* term trees but also *threshold* allowing to select highly similar term trees. **approximateTreeMatch** fills *result* array (line 3) which stores the mappings holding between nodes of the trees. *approximationScore* is computed (line 4) by **analyzeMismatches**. If *approximationScore* exceeds *threshold* the mappings calculated by **approximateTreeMatch** are returned (line 6). In contrast to **exactTreeMatch** presented in Figure 7 **approximateTreeMatch** considers also semantic relations other then equivalence (line 13) and stores them in *result* array (line 14). **analyzeMismatches** decides the importance of the mismatches among the nodes of the trees (if any) and calculates the aggregate score of tree match quality by exploiting a tree edit distance algorithm [30, 28].

```
1.MappingElement[] approximateStructureMatch(
  Tree of Nodes source, target, double threshold)
2. MappingElement[] result;
3. approximateTreeMatch(source,target,result);
4. double approximationScore=analyzeMismatches(source,
                              target,result);
5. if (approximationScore>threshold)
6.  return result;
7. else
8.  return null;

9. void approximateTreeMatch(Tree of Nodes source,
                    target,MappingElement[] result)
10. Node sourceRoot=getRoot(source);
11. Node targetRoot=getRoot(target);
12. String relation= nodeMatch(sourceRoot,targetRoot);
13. if (relation!="Idk")
14.  addMapping(result,sourceRoot,targetRoot,relation);
15. Node[] sourceChildren=getChildren(sourceRoot);
16. Node[] targetChildren=getChildren(targetRoot);
17. For each sourceChild in sourceChildren
18.  Tree of Nodes sourceChildSubTree=
                              getSubTree(sourceChild);
19.  For each targetNode in target
20.   Tree of Nodes targetChildSubTree=
                              getSubTree(targetChild);
21.   approximateTreeMatch(sourceChildSubTree,
                    targetChildSubTree, nodesToMatch);
```

Figure 9: Pseudo Code for Approximate Structural Matching

Let us give a brief (due to a lack of space) description of the tree edit distance problem. In its traditional formulation, the tree edit distance problem considers three operations: (a) vertex removal, (b) vertex insertion, and (c) vertex replacement [28]. To each of these operations, a cost is assigned. The solution of this problem consists in determining the minimal set of operations (i.e., the one with the minimum cost) to transform one tree into another. Another equivalent (and possibly more intuitive) formulation of this problem is to discover a (proper) mapping with minimum cost between the two trees. The concept of (proper) mapping (introduced in [28]) is defined next.

Let T_x be a tree and let $T_x[i]$ be the i-st vertex of tree T_x in a preorder walk of the tree. A (proper) mapping between a tree T_1 of size n_1 and a tree T_2 of size n_2 is a set M of ordered pairs (i, j), satisfying the following conditions for all $(i_1, j_1), (i_2, j_2) \in M$:

1. $i_1 = i_2$ iff $j_1 = j_2$;
2. $T_1[i_1]$ is on the left of $T_1[i_2]$ iff $T_2[j_1]$ is on the left of $T_2[j_2]$;
3. $T_1[i_1]$ is an ancestor of $T_1[i_2]$ iff $T_2[j_1]$ is an ancestor of $T_2[j_2]$.

In our case the approximate structure matching algorithm produces a partial mapping among the nodes of two trees. In order to apply a tree edit distance algorithm we have to ensure that it is a (proper) mapping (i.e, it satisfies the conditions presented above). The first condition requires a 1 to 1 mapping. Therefore we drop from the partial mapping all the correspondences that violate this requirement. The second condition requires the order preservation among sibling nodes. Notice that sibling ordering does not influence on the ability of the peer to interpret the constraint. Therefore, in order to satisfy the condition, the sibling nodes have to be reordered. Figure 11 illustrate an example of such reordering for the trees depicted on Figure 10. The third condition enforces the hierarchical relation between the nodes of the trees. In order to satisfy it we drop from the partial mapping all the correspondences that violate this condition. Notice that the approximate structure

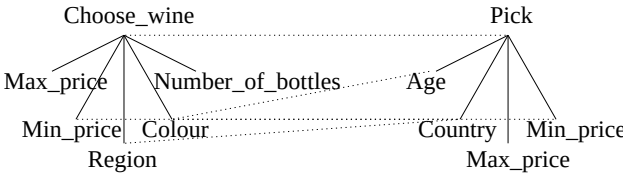


Figure 10: Approximate mappings between two constraints

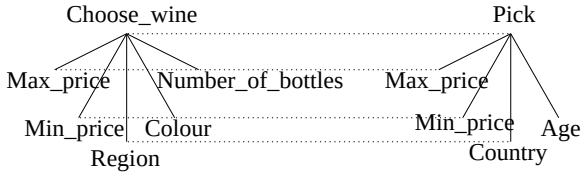


Figure 11: Reordering of constraints

matching algorithm produces the correspondences which stand for equivalence, less/more generality and disjointness relations. In order to discriminate among them we modified the costs of the tree edit distance operations as depicted in Table 1.

Table 1: Cost of tree editing operations

Operation	Cost
replace(a,b), $a = b$	0
replace(a,b), $a \sqsubseteq b$	0.5
replace(a,b), $a \sqsupseteq b$	0.5
replace(a,b), $a \perp b$	∞
replace(a,b), a is not related to b	1
insert(a)	1
delete(a)	1

To ensure a quick prototyping approach we selected a simple tree edit distance algorithm from Valiente's work [29]. In this algorithm deletion and insertion operations are performed only on the leave nodes. When deleting a non-leaf node v, every node in the subtree rooted at v has to be deleted first. The same applies to the insertion of non-leaf nodes. The algorithm finds the least-cost transformation of an ordered tree T_1 and T_2 in $O(|n_1||n_2|)$ time using $O(|n_1||n_2|)$ additional space (see Lemma 2.20 in [29]).

Since we were interested in similarity rather than in distance we exploited the following similarity score:

$$Sim = 1 - \frac{EditDistance}{max(n_1, n_2)} \quad (1)$$

where n_1 and n_2 stand for the number of nodes in the trees.

EXAMPLE 5. *(Approximate Matching in the Interaction Model) Imagine if the classification that the $customer$ peer was attempting to match $choose\_wine(P_1, P_2, R, C, N)$ to was in fact, $pick(Ag, Ct, P_1, P_2)$, where N, P_1 and P_2 represent the same types as previously, Ag represents type age and Ct represents type country. The mappings between these terms are illustrated in Figure 10. The map between the root node would be discovered as explained in Example 3. The first node at the second level in the left-hand tree, Max_price would, as before, find an exact match, as would Min_price. The node $Region$ has a map to $Country$, as, according to the peer's taxonomy, a region is part of a country (i.e, $Region \sqsubseteq Country$). Notice that Colour, Number of Bottles and Age nodes are left unmapped. Therefore the edit distance between the trees given the weights for the editing operations described above is: 0.5+1+1=2.5. The similarity score for the trees depicted on Figure 10 is 1-2.5/6=0.58.*

5. EVALUATION

We have implemented the algorithm described in the previous section in Java and evaluated its matching quality on the 66 pairs of similar first order logic terms extracted from different versions of the Standard Upper Merged Ontology (SUMO)[1] and the Advance Knowledge Transfer (AKT)[2] ontologies. We extracted all the differences between versions 1.50 and 1.51, and 1.51 and 1.52 of the SUMO ontology and between versions 1, 2.1 and 2.2 of the AKT-portal and AKT-support ontologies[3]. These are both first-order ontologies, so many of these differences mapped well to the potential differences between constraints that we are investigating. However, some of them were more complex, such as differences in inference rules, or consisted of ontological objects being added or removed rather than altered, and had no parallel in our work. These pairs of terms were discarded and our tests were run on all remaining differences between these ontologies. We have therefore simulated the situation when the peer capabilities are defined in one version of the ontology and the constraints in the interaction model are expressed exploiting the other version of the same ontology.

We have calculated Recall defined as the ratio of the correct correspondences produced by the matching system to the total number of correct correspondences. We have also calculated the Recall for the various values of threshold for approximate structure matching algorithm. Since all the pairs of terms in the dataset are equivalent we were unable to calculate Precision defined as the ratio of correct correspondences to all the correspondences produced by the matching system. The evaluation was performed on the Pentium 4 computer.

Interestingly enough our exact structure matching algorithm was able to find 36 correct correspondences what stands for 54% of Recall. All mismatches corresponded to structural differences among first order terms which exact structure matching algorithm is unable to capture.

The examples of correctly found correspondences are given below:

```
meeting-attendees(has-other-agents-involved)
meeting-attendee(has-other-agents-involved)

r&d-institute(Learning-centred-organization)
r-and-d-institute(Learning-centred-organization)

piece(Pure2,Mixture)
part(Pure2,Mixture)

has-affiliatied-people(Affiliated-person)
has-affililated-person(affiliated-person)
```

The first and the second example illustrate the minor syntactic differences which prevent the unification process to interpret the term correctly, while the third and fourth examples illustrate the semantic heterogeneity in the various versions of the ontologies.

Figure 12 presents the Recall of approximate structure matching algorithm depending on the cut-off threshold value. As from the figure the algorithm demonstrates high Recall on the wide range of threshold values. For example, Recall for 0.5 threshold is slightly lower than 88% what is a high result for any of state of the art

[1]http://ontology.teknowledge.com/

[2]http://www.aktors.org

[3]see http://dream.inf.ed.ac.uk/projects/dor/ for full versions of these ontologies and analysis of their differences

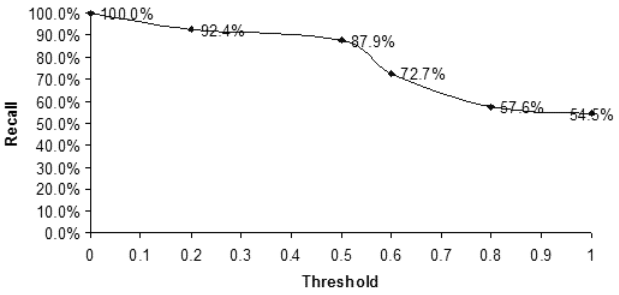


Figure 12: Recall depending on threshold value

matching systems (see [6] for the latest ontology matching evaluation results).

Table 2 summarizes the time performance of the matching algorithm. It presents the average time taken by the various steps of

Table 2: Time performance of approximate structure matching algorithm (average on 66 term matching tasks)

	Node matching Step 1 and 2	Node matching Step 3 and 4	Tree matching
Time, ms	158.6	4.8	1.2

the algorithm on 66 term matching tasks. As from the table Step 1 and 2 of the node matching algorithm significantly slow down the whole process. However, as discussed in section 4.1, these steps correspond to the linguistic preprocessing of the natural language labels that can be performed once and later reused by the matching process. Given that the term can be automatically annotated with the linguistic preprocessing results [9], the term matching task is performed in average in 6 ms what corresponds roughly to 160 term matching tasks per second and satisfies to the requirements of run time web service composition.

6. RELATED WORK

The main contribution of the work presented in this paper is to provide an alternative approach to Web service composition. The state of the art approaches to the web service composition either rely on the manual composition in design time [2] or provide the support for (semi-)automated web service composition given that the web services are annotated with the concepts derived from either shared ontology or vocabulary [7, 18]. Our approach is in between of these two extremes. The interaction models described in this paper differently from [2] do not specify the services participating in the interaction but define the constraints the services have to satisfy. This kind of specification on the other hand allows us to simplify the problem of run time web service composition and relax a shared ontology requirement.

In particular, Web service composition techniques [7, 18] tend to assume that (i) services are annotated with terms taken from an ontology; (ii) this ontology is shared between multiples services or between query and services; (iii) this ontology is semantically rich enough to do inference. In our approach we assume that: (i) services are annotated with first order terms; (ii) the terms are not standardized between services; (iii) they are not embedded in a structure on which to do inference. Additionally the standard web service composition approaches assume a predefined recruitment of who will run what services, which depends on large amounts of pre-interaction organization and is often not practical, whereas we allow run-time recruitment for services. Moreover, standard approaches allow run-time composition of simple services into complex ones whereas we use predefined interaction models to define the ways in which composition can occur. This restriction - of recruiting services for predefined workflows rather than performing arbitrary composition - means that many of the difficulties that have proved so far insurmountable become tractable.

The problem of location of web services on the basis of the capabilities that they provide (often referred as matchmaking problem) recently have received a considerable attention. Most of the approaches to the matchmaking problem so far employed a single ontology approach (i.e., the web services are assumed to be annotated with the concepts taken from the shared ontology). See [24, 14, 15] for example. Probably the most similar to ours was the approach taken in [23] where the services are assumed to be annotated with the concepts taken from various ontologies. Then the matchmaking problem is solved by the application of the matching algorithm. The algorithm combines the results of atomic matchers that roughly correspond to the element level matchers exploited by us in the Step 3 of node matching algorithm in Section 4.1. In contrast to this work we exploit a more sophisticated matching technique that allow us to utilize various forms of context.

The ontology matching problem has received a considerable attention in the last years. Many diverse solutions have been proposed so far [21, 16, 5, 26, 3]. However most of efforts were devoted to computation of the correspondences holding among the classes of description logic ontologies. Recently several approaches allowed computation of correspondences holding among the object properties (or binary predicates) [13, 20, 27]. Differently from these approaches we allow computation of correspondences holding among first order terms.

7. CONCLUSION

The work described in this paper constitutes a new approach to web service composition: the idea that through the use of shared declarative interaction specifications, which can be discovered during run-time, we can facilitate automatic interaction between services without enforcing strict semantic agreement or annotation. We presented the techniques through which this idea can be implemented. We extended the established LCC language to allow for the inclusion of contextual information to facilitate semantic matching. We then developed novel techniques that allow existing semantic matching techniques to be applied to the matching of logical terms through consideration of the structure of those terms. The combination of these two contributions extend the applicability of these specifications of interactions from a closed to an open environment, and are thus applicable to peer-to-peer networks of arbitrary size. Finally, we introduced our approximate matching techniques, which extend our semantic matching techniques to allow the return of good enough answers in the case where perfect answers are unavailable.

8. ACKNOWLEDGMENT

This work has been supported by the STReP OpenKnowledge (http://www.openk.org/).

9. REFERENCES

[1] P. Avesani, F. Giunchiglia, and M. Yatskevich. A large scale taxonomy mapping evaluation. In *Proceedings of International Semantic Web Conference (ISWC)*, pages 67–81, 2005.

[2] T. Andrews, F. Curbera, H. Dolakia, J. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weeravarana. Business Process Execution Language for Web Services (version 1.1)., 2003.

[3] S. Castano, A. Ferrara, and S. Montanelli. Matching ontologies in open networked systems: Techniques and applications. *Journal on Data Semantics (JoDS)*, V, 2005.

[4] D.Robertson, C.Walton, A.Barker, P.Besana, Y.Chen-Burger, F.Hassan, D.Lambert, G.Li, J. McGinnis, N.Osman, A.Bundy, F.McNeill, F. van Harmelen, C.Sierra, and F.Giunchiglia. Models of interaction as a grounding for peer to peer knowledge sharing. *Advances in Web Semantics*, 1, (in press).

[5] M. Ehrig, S. Staab, and Y. Sure. Bootstrapping ontology alignment methods with APFEL. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 186–200, 2005.

[6] J. Euzenat, M. Mochol, O. Svab, V. Svatek, P. Shvaiko, H. Stuckenschmidt, W. van Hage, and M. Yatskevich. Introduction to the ontology alignment evaluation 2006. In *Proceedings of Ontology Matching 2006 Workshop at ISWC'06*, 2006.

[7] D. Fensel and C. Bussler. The web service modelling framework. *Electronic commerce: Research and applications*, pages 1:113–137, 2002.

[8] F. Giunchiglia and P. Shvaiko. Semantic matching. *The Knowledge Engineering Review*, 18(3):265–280, 2003.

[9] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-Match: an algorithm and an implementation of semantic matching. In *Proceedings of the European Semantic Web Symposium (ESWS)*, pages 61–75, 2004.

[10] Fausto Giunchiglia. Contextual reasoning. *Epistemologia, special issue on I Linguaggi ele Macchine*, XVI:345–364, 1993.

[11] Fausto Giunchiglia and Ilya Zaihrayeu. Making peer databases interact - a vision for an architecture supporting data coordination. In *Proceedings of the International Workshop on Cooperative Information Agents (CIA)*, pages 18–35, 2002.

[12] P. Haase, R. Siebes, and F. van Harmelen. Expertise-based peer selection in peer-to-peer networks. *Knowledge and Information Systems*, 2006.

[13] J. Kim, M. Jang, Y. Ha, J. Sohn, and S. Lee. MoA: OWL ontology merging and alignment tool for the semantic web. In *Proceedings of the the International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE*, pages 722 – 731, 2005.

[14] M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with owls-mx. In *AAMAS*, pages 915–922, 2006.

[15] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 331–339, New York, NY, USA, 2003. ACM Press.

[16] V. Lopez, M. Sabou, and E. Motta. Powermap: Mapping the real semantic web on the fly. In *The Semantic Web - ISWC 2006*, pages 414–427, 2006.

[17] B. Magnini, M. Speranza, and C. Girardi. A semantic-based approach to interoperability of classification hierarchies: Evaluation of linguistic techniques. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, 2004.

[18] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara. Bringing semantics to web services: The owl-s approach. In *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, pages 26–42, San Diego, CA, USA, July 6-9 2004.

[19] G. Miller. WordNet: A lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[20] P. Mitra, N. Noy, and A. Jaiswal. OMEN: A probabilistic ontology mapping tool. In *Proceedings of the Meaning Coordination and Negotiation Workshop at the International Semantic Web Conference (ISWC)*, 2004.

[21] N. Noy and M. Musen. Anchor-PROMPT: Using non-local context for semantic matching. In *Proceedings of the Workshop on Ontology and Information Sharing at the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 63–70, 2001.

[22] OpenKnowledge Manifesto. http://www.openk.org/, 2006.

[23] S. Oundhakar, K. Verma, K. Sivashanugam, A. Sheth, and J. Miller. Discovery of web services in a multi-ontology and federated registry environment. *International Journal of Web Services Research (JWSR)*, 2(3):1–32, 2005.

[24] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 333–347, 2002.

[25] H. Syed, P. Andritsos. Weigh your Preferences! Towards using hierarchies for building personal libraries. Technical Report, University of Trento, 2006.

[26] Y. Qu, W. Hu, and G. Cheng. Constructing virtual documents for ontology matching. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 23–31, New York, NY, USA, 2006. ACM Press.

[27] U. Straccia and R. Troncy. oMAP: Combining classifiers for aligning automatically owl ontologies. In *Proceedings of the International Conference on Web Information Systems Engineering (WISE)*, pages 133–147, 2005.

[28] K. Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.

[29] G. Valiente. *Algorithms on Trees and Graphs*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[30] K. Zhang, R. Statman, and D. Shasha. On the editing distance between unordered labeled trees. *Inf. Process. Lett.*, 42(3):133–139, 1992.